(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 0 766 172 B1

(12) **EUROPEAN PATENT SPECIFICATION**

(45) Date of publication and mention of the grant of the patent:
17.10.2001 Bulletin 2001/42

(51) Int Cl.[7]: **G06F 9/46**

(21) Application number: **96305456.4**

(22) Date of filing: **25.07.1996**

(54) **A method and apparatus for allowing generic stubs to marshal and unmarshal data in object reference specific data formats**

Verfahren und Vorrichtung zum Verpackung und Entpackung von Daten in objectreferenzspezifischen Datenformaten anhand generischen Stubs

Procédé et appareil pour permettre des "stubs" génériques de transformer et rétablir des données dans des formats de données spécifiques

(84) Designated Contracting States:
DE FR GB IT NL

(30) Priority: **28.09.1995 US 534966**

(43) Date of publication of application:
02.04.1997 Bulletin 1997/14

(73) Proprietor: **SUN MICROSYSTEMS, INC.**
Mountain View, CA 94043 (US)

(72) Inventors:
- **Hamilton, Graham**
  Palo Alto, California 94303 (US)
- **Lim, Swee Boon**
  Mountain View, California 94043 (US)
- **Kessler, Peter B.**
  Palo Alto, California 94306 (US)
- **Nisewanger, Jeffrey D.**
  San Jose, California 95112 (US)
- **Radia, Sanjay R.**
  Fremont, California 94555 (US)

(74) Representative: **W.P. Thompson & Co.**
Coopers Building,
Church Street
Liverpool L1 3AB (GB)

(56) References cited:
EP-A- 0 501 610 **EP-A- 0 604 010**
EP-A- 0 643 349

- **OS/2 DEVELOPER, vol. 6, no. 5, September 1994 - October 1994, SAN FRANCISCO, US, pages 46-53, XP002022570 ROBERT ORFALI AND DAN HARKEY: "Client/Server Programming with CORBA Objects"**

EP 0 766 172 B1

Description

[0001] The present invention relates to the fields of distributed computing systems, client-server computing and object oriented programming.

[0002] Specifically, the present invention relates to a method and apparatus for providing program mechanisms which allow generic stubs to marshal and unmarshal data in object-reference-specific data formats.

[0003] A key problem in modern object oriented distributed processing systems is permitting object applications to communicate with a new object request broker ("ORB") which has its own unique data format, with only minor modification to the supporting code mechanisms. Similarly client applications and stubs should be able to communicate seamlessly with different ORBs in a system which can communicate with a multiplicity of ORBs wherein each of the ORBs has its own unique data format.

[0004] For example, the Object Management Group (OMG) is an industry standards organization that is creating multi-vendor standards for distributed object-oriented programming. One of the cornerstones of the OMG work has been the definition of a common Interface Definition Language, known as IDL, that is now in widespread use as a standard way for defining interfaces to network objects in a way that is independent of the particular network protocols that are being used or the particular programming languages that are being used by the clients and the servers.

[0005] The IDL language is object-oriented and supports multiple inheritance. It comes with a rich set of built-in primitive types (such as floats, booleans, etc.) and also defines a set of structured types including structs, unions and sequences.

[0006] The Object Management Group standardized on the IDL interface definition language as a uniform way of defining interfaces to network objects. However, the OMG initially left the on-the-wire protocol and data formats undefined. As a result different vendors implemented ORBs with different protocols and different data formats.

[0007] Recently, the OMG has agreed on a common ORB inter-operability protocol, the Universal Networked Objects (UNO) protocol. However this is primarily viewed as a gateway protocol for connecting object systems from different vendors. At least in the short term different vendors appear likely to continue to use their existing protocols for higher performance within their own object systems, while supporting lower performance UNO gateways to other ORBs. Thus a current need exists to permit object applications to transparently communicate with these ORBs with different protocols and different data formats.

[0008] For further description of object oriented design and programming techniques see "Object-oriented Software Construction" by Bertrand Meyer, Prentice-Hall 1988. For further information on OMG, CORBA, ORBs and IDL see the "Common Object Request Broker: Architecture and Specification", Revision 2.0, July 1955.

[0009] Currently Internet browsers are very limited in their ability to interact with network servers. Typically a browser such as Mosaic will down-line load an HTML document and then wait passively for a human user to either select another document or to enter HTML forms information that can be passed back to the server.

[0010] Two new developments are promising to make the Internet a more dynamic environment. The first is support for scripting languages in Internet browsers (such as the Sun Microsystems, Inc. (SUN) JAVA language described below) so that a browser can download and execute interactive scripts. This transforms browsers from being passive viewers into being dynamic agents that can interact with the Internet on a user's behalf and display rapidly changing information. The second development is the widespread adoption of the Object Management Group's distributed object interfaces based on the IDL interface definition language. These provide a standard way for network servers to export services as "network objects" in a language and vendor independent way. This will make it much easier to build network services that can be transparently accessed from different client environments.

[0011] EP-0604010 discloses an apparatus and a method comprising a logic module called a sub-contract, that has been designed to provide control of the basic mechanisms of object invocation and argument passing that are most important in distributed systems, in a way which makes it easy for object implementors to select and use an existing sub-contract, and which permits the application programmers to be unaware of the specific sub-contracts that are being used for particular objects.

[0012] It includes a new type of object, termed a "spring object", which includes a method table, a subcontract mechanism and a data structure which represents the subcontract's local private state.

[0013] Each subcontract contains a client-side portion and a related server-side portion. Each object type has an associated subcontract The client-side portion of a subcontract has the ability to generate a new spring object, to delete a spring object, to marshal information about its associated object into a communications buffer, to unmarshal data in a communications buffer which represents its associated object, to transmit a communications buffer to its associated server-side subcontract, which includes either transmitting an object from one location to another or transmitting a copy of an object from one location to another. The server-side portion of the subcontract mechanism includes the ability to create a spring object, to provide support for processing incoming calls and related communications buffers and to provide support for revoking an object.

[0014] It also includes methods for using subcontracts to process object invocations, including the passing of

arguments, which arguments may themselves be objects, in a distributed computing system, wherein the client applications may be on different computers from the object implementations.

[0015] EP-0643349 discloses an apparatus and a method comprising logic modules, called a client-side stub generator, a database of compressed client-side stub execution code and a client-side stub interpreter, that have been designed to minimize the memory space required by client-side stubs while retaining the design of stubs to provide control of the basic mechanisms of object invocation and argument passing that are most important in distributed systems, and which permits the application programmers to be unaware of the specific stubs that are being used for particular objects.

[0016] The mechanism used to reduce this memory space comprises a stub generator (called "CONTOCC") for generating a data base of compressed client-side stub description filed which are stored in computer memory and a stub-interpreter which knows how to read these client-side stub description files. CONTOCC reads interface definition language ("IDL") files and generates corresponding C++ files. CONTOCC has the ability to read the IDL data and generate either uncompressed C++ stub files or the special compressed client-side stub interpreter files which contain only those data that are unique to the particular stub method involved in byte coded form.

[0017] The stub interpreter logic module receives a target object, arguments related to the particular stub method called and a pointer to the compressed byte code representations of the stub code in the data base of compressed client-side stub description files which are stored in computer memory, and proceeds to execute the object invocation, marshaling or unmarshaling of any return messages as required.

Java

[0018] Java is a strongly typed object-oriented language with a C like syntax. The Java compiler and runtime code mechanisms enforce type safety so that there can be no wild pointers or other references that violate the language's type system. So for example, there is no "void*" and all casts are validated at runtime.

[0019] The Java language is typically compiled to machine-independent byte-codes and then a Java virtual machine interprets those byte codes in order to execute the Java program. Java can be integrated into network HTML browsers, so that as part of viewing a document one can down-line load a set of Java byte-codes and then execute them on the client machine. Because Java is completely typesafe the client browser can feel confident that the Java program can be executed safely without endangering the security or integrity of the client Java is more fully described in "The Java Language Specification" release 1.0 Alpha3, by Sun Microsystems, Inc. dated May 11, 1995.

[0020] Scripted language systems like Java generate Java programs that are designed to be portable and to be deployed in a variety of different environments. It is therefore desired to allow Java programs to use different ORBs without requiring any changes to the Java program. Because the generated stubs are part of the Java program it is necessary that the stubs be ORB-independent so that the Java program and its associated stubs might be used in any of a multiplicity of ORBs.

[0021] This disclosure describes a solution to the basic problem by creating a generic interface between the stubs and ORB specific data mechanisms. These ORB specific data mechanisms include one or more MarshalBuffer Objects which have methods for marshaling and unmarshaling one or more particular ORB related on-the-wire data formats and a method and apparatus for using an object reference (Objref) to indicate the particular MarshalBuffer Object to use for this particular Object call.

[0022] The present invention provides an elegant and simple way to provide mechanisms for invocation of objects by client applications and for argument passing between client applications and object implementations, without the client application or the operating system knowing the details of how these mechanisms work. Moreover, these mechanisms functions in a distributed computer environment with similar ease and efficiency, where client applications may be one computer node and object implementations on another.

[0023] In one aspect of the invention, there is provided a computer system having a processor, a memory, a display device, an input/output device, and an operating system (OS) for processing program code mechanism invocations which are directed to one of a multiplicity of remote Object Request Brokers (ORBs), characterised in that the computer system further comprises:

an ORB independent layer of code mechanisms including program applications and related stubs; and
an ORB dependent layer of code mechanisms, coupled to the ORB independent layer, the ORB dependent layer comprising

program code mechanisms which are ORB-specific and which are configured to generate a marshal code mechanism to marshal data in an ORB required format, and
program code mechanisms configured to use a specific network protocol code whereby a program application code mechanism from the ORB independent layer can invoke a call on a remote implementation program code mechanism and the ORB-specific program code mechanisms in the ORB dependent layer will determine an appropriate marshal code mechanism to marshal data in an ORB required format;
thereby providing a computer system capable

of communicating with a multiplicity of ORBs wherein each ORB requires data to be in a specific format.

[0024] In another aspect of the invention there is provided a method of operating a computer system having a processor, a memory, a display, an input/output mechanism, an operating system and at least one client application program, one or more stub programs related to the client application, the method performed by the computer being characterised by the steps of:

invoking a call on a program implementation code mechanism, the call invocation being made by a client application wherein the call includes a reference to the program implementation code mechanism;
using a stub program code mechanism which is related to the client application to receive the call invocation, wherein the stub program code mechanism has no knowledge of a format required for marshaling data provided by the client application in connection with the call invocation;
calling a first specific program code mechanism by the stub program code mechanism, requesting the first specific program code mechanism to provide a MarshalBuffer code mechanism that knows how to format data provided by the client application in connection with the call invocation;
marshaling the data provided by the client application in connection with the call invocation, the marshaling being done by the stub program code mechanism using the MarshalBuffer code mechanism; and
sending the call invocation to a server containing the program implementation code mechanism which is the target of the call.

[0025] In yet another aspect of the invention there is provided a computer program product comprising a computer system usable storage medium having computer readable code embodied therein for causing a computer system to process program code mechanism invocations which are directed to one of a multiplicity of remote Object Request brokers (ORBs), the computer program product being characterised by:

a first computer readable program code mechanism configured to comprise an ORB-independent layer of code mechanisms including client applications and related stub program code mechanisms; and
a second computer readable program code mechanism configured to comprise an ORB dependent layer of code mechanisms coupled to the ORB independent layer, the ORB dependent layer comprising program code mechanisms which are ORB-specific and which are configured to generate a marshal code mechanism to marshal data in an ORB required format and program code mechanisms configured to use a specific network protocol code whereby a program application code mechanism from the ORB independent layer can invoke a call on a remote implementation program code mechanism and the ORB-specific program code mechanisms in the ORB dependent layer will determine an appropriate marshal code mechanism to marshal data in an ORB required format, thereby providing a computer system capable of communicating with a multiplicity of ORBs wherein each ORB requires data to be in a specific format.

[0026] The present invention will now be further described, by way of example, with reference to the accompanying drawings, in which:-

Figure 1 illustrates the configuration of a typical computer hardware system used with and as a part of the present invention.
Figure 2 illustrates the prior art relationship of client and server applications to stubs and network software.
Figure 3 illustrates a system showing Java ORB classes.
Figure 4 illustrates the relationship between stub objects, ORB specific code/subcontract mechanisms, and server application objects in a single ORB system.
Figure 5 illustrates remote object invocation using ORB specific code/subcontracts, ORB-specific MarshallBuffer mechanisms in a multi-ORB system.
Figure 6 illustrates an exemplary MarshallBuffer code mechanism.
Figure 7 illustrates a more specific remote object invocation using ORB specific code/subcontracts, ORB-specific MarshallBuffer mechanisms in a multi-ORB system.

NOTATIONS AND NOMENCLATURE

[0027] The detailed descriptions which follow may be presented in terms of program procedures executed on a computer or network of computers. These procedural descriptions and representations are the means used by those skilled in the art to most effectively convey the substance of their work to others skilled in the art.

[0028] A procedure is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like. It should be noted, however, that all of these and similar terms are to be associated with the appro-

priate physical quantities and are merely convenient labels applied to these quantities.

[0029] Further, the manipulations performed are often referred to in terms, such as adding or comparing, which are commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein which form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or similar devices.

[0030] The present invention also relates to apparatus for performing these operations. This apparatus may be specially constructed for the required purposes or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The procedures presented herein are not inherently related to a particular computer or other apparatus. Various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description given.

[0031] The following disclosure describes solutions to the problems which are encountered by object oriented systems designers when attempting to implement schemes for object invocation and for argument passing in distributed systems wherein the arguments may be objects, in ways which do not lock the object oriented base system into methods which may be difficult to change at a later time. Moreover, the invention disclosed describes a "***MarshalBuffer mechanism***" which contains operations (called "*methods*" in Object Oriented programming parlance) for marshaling data for a specific ORB; a "***Multi-ORB Marshaling system***" which permits a client application and related stub to invoke an operation on a target object without any knowledge or concern about which ORB this target object uses or what data formate the ORB requires for the arguments of the operation invoked; and a "***Computer system for multi-ORB communication***" comprising an ORB independent layer which contains client applications and stubs; an ORB dependent-OS independent layer which contains ORB dependent code/Subcontract code mechanisms as well as ORB specific MarshalBuffers for a multiplicity of ORBs; and an Operating System (OS) layer. The "***ORB dependent code mechanism***" used in the present invention, is analogous to the "***subcontract mechanism***" which is associated with each object and which is described in EP-A-0 604 010.

[0032] In the following description, for purposes of explanation, specific data and configurations are set forth in order to provide a thorough understanding of the present invention. The preferred embodiment described herein is implemented as a portion of the SPRING-DISTRIBUTION Object-Oriented Operating System created by Sun Microsystems®, Inc. (Sun Microsystems is a registered trade mark of Sun Microsystems, Inc.). However, it will be apparent to one skilled in the art that the present invention may be practised without the specific details and may be implemented in various computer systems and in various configurations, or makes or models of tightly-coupled processors or in various configurations of loosely-coupled multi processor systems. The Spring-Distribution Object-Oriented Operating System is described in "A Spring Collection" A Collection of Papers on the Spring distributed Object-Oriented operating System published September 1994 by Sun Microsystems, Inc..

Operating Environment

[0033] The environment in which the present invention is used encompasses the general distributed computing system, wherein general purpose computers, workstations, or personal computers are connected via communication links of various types, in client-server arrangement, wherein programs and data, many in the form of objects, are made available by various members of the system for execution and access by other members of the system. Some of the elements of a general purpose workstation computer are shown in Figure 1, wherein a processor 1 is shown, having an Input/Output ("I/O") section 2, a central processing unit ("CPU") 3 and a memory section 4. The I/O section 2 is connected to a keyboard 5, a display unit 6, a disk storage unit 9 and a CD-ROM drive unit 7. The CD-ROM unit 7 can read a CD-ROM medium 8 which typically contains program code mechanisms 10 and data.

Stubs

[0034] Techniques for providing a language-level veneer for remote operations (for example, "Remote Procedure Calls") have been in use for many years. Typically these take the form that a remote interface is defined in some language. Then a pair of stubs are generated from this interface. The client stub runs in one machine and presents a language level interface that is derived from the remote interface. The server stub runs in some other machine and invokes a language-level interface that is derived from the remote interface. Referring now to Figure 2, to perform a remote operation, a client application 12 on one machine 11, invokes the chent stub 14, which marshals the arguments associated with the invocation into network buffer(s) and transmits them to the server stub 22 on the remote machine 18, which *unmarshals* the arguments from the network buffer(s) and calls the server application 24. Similarly, when the server application 24 returns a response, the results are marshaled up by the server stub 22 and returned to the client stub 14, which unmarshals the results and returns them to the client application 12. The

entire mechanics of argument and result transmission, and of remote object invocation, are performed in the stubs. Both the client application and the server application merely deal in terms of conventional language-level interfaces.

[0035] When the arguments or results are simple values such as integers or strings, the business of marshaling and unmarshaling is reasonably straightforward. The stubs will normally simply put the literal value of the argument into the network buffer. However, in languages that support either abstract data types or objects, marshalling becomes significantly more complex. One solution is for stubs to marshall the internal data structures of the object and then to unmarshal this data back into a new object This has several serious deficiencies. First, it is a violation of the "abstraction" principle of object-oriented programming, since stubs have no business knowing about the internals of objects. Second, it requires that the server and the client implementations of the object use the same internal layout for their data structures. Third, it may involve marshalling large amounts of unnecessary data since not all of the internal state of the object may really need to be transmitted to the other machine. An alternative solution is that when an object is marshalled, none of its internal state is transmitted. Instead an identifying token is generated for the object and this token is transmitted. For example in the Eden system, objects are assigned names and when an object is marshalled then its name rather than its actual representation is marshalled. Subsequently when remote machines wish to operate on this object, they must use the name to locate the original site of the object and transmit their invocations to that site. This mechanism is appropriate for heavyweight objects, such as files or databases, but it is often inappropriate for lightweight abstractions, such as an object representing a cartesian coordinate pair, where it would have been better to marshal the real state of the object. Finally, some object-oriented programming systems provide the means for an object implementation to control how its arguments are marshalled and unmarshalled. For example, in the Argus system object implementors can provide functions to map between their internal representation and a specific, concrete, external representation. The Argus stubs will invoke the appropriate mapping functions when marshalling and unmarshaling objects so that it is the external representation rather than any particular internal representation that is transmitted. These different solutions all either impose a single standard marshalling policy for all objects, or require that individual object implementors take responsibility for the details of marshalling. An advanced object marshaling process is described in the above referenced EP-A-0 604 010 which describes "**Subcontracts.**"

Current Problem Summary

Specific Problem for Current Object Oriented Systems

[0036] The specific problem here is that different ORBs have different on-the-wire data formats. So one ORB might marshal bytes in little-endian order, another in big-endian, etc. Different ORBs also have different on-the-wire formats for arrays, strings, unions, etc.

[0037] So it is desirable to have ORB independent stubs that can marshal their data differently when talking to different objects. Thus, one might use a DEC data format when talking to a DEC object and a Sun data format when talking to a Sun object. And it is desirable for one to allow a single set of stubs to be in simultaneous use with different ORBs.

Proposed Solution to the Problem

[0038] The solution is;

(1) define a generic interface for marshalling. This generic interface provides methods for marshalling and unmarshalling ints, shorts, bytes, chars, and strings. But it says nothing about how these methods are implemented.

(2) Different ORBs provide their own implementation of the generic marshal buffer interface. As well as supporting the generic marshalling and unmarshalling methods, these ORB specific marshal classes may provide additional methods for marshaling ORB specific data. For example the Spring ORB implementation of MarshalBuffer provides methods for marshalling and unmarshalling Spring doors.

(3) Each object reference (Objref) contains a pointer to a set of ORB runtime machinery that belongs to the ORB that implements that object. In the preferred implementation this consists of a pointer to the client side ORB-dependent code mechanism/subcontract for the object.

(4) The ORB runtime machinery described in (3) will support a method for obtaining a MarshalBuffer object. The runtime machinery will return a MarshalBuffer object that implements the correct marshalling and unmarshalling for that ORB.

(5) The generic stubs work entirely in terms of the generic MarshalBuffer interface.

(6) At the beginning of each call, the generic stubs call into the ORB specific runtime code mechanisms associated with the object reference to get an appropriate MarshalBuffer object. The stubs

then use the generic marshalling and unmarshalling interfaces to marshal and unmarshal data to and from that MarshalBuffer object. Since the implementations of these marshal and unmarshal methods are ORB specific, this means that the data is being marshalled and unmarshalled in an ORB specific way.

[0039] In the currently preferred embodiment, the generic MarshalBuffer includes additional capabilities for handling differences in several known ORBs. For example,
(a) in addition to methods for marshalling and unmarshalling simple data types, the generic MarshalBuffer provides a way for Marshalling array descriptors. This method takes the bounds of the array and then marshals an array descriptor in an ORB specific way. For example, the DOE ORB code marshals the length of the array. But the Spring ORB code marshals the lower bound and the upper bound, and (b) similarly provides mechanisms for unmarshalling an array descriptor.

HOW TO MAKE AND USE THE INVENTION

A Portable ORB Implementation

[0040] One of the goals for the preferred embodiment of the present invention is for the Java ORB implementation to be allowed to work directly with a variety of different on-the-wire protocols and data formats. In particular, a single Java program must be able to simultaneously use object references that refer to objects in different ORBs. The internet is a very heterogeneous environment and it is desired not to restrict Java IDL clients to only working with a single server at a time.

[0041] In particular the Java ORB implementation of the preferred embodiment must be able to communicate directly with both Sun's Distributed Object Environment (DOE)and with the Spring distributed operating system. It is also desired to design the Java ORB core so that it could communicate with UNO gateways or with DCE based object systems.

Portability issues

[0042] Portability is an issue at several different levels.

[0043] At the lowest level, Java's socket class is used to get machine and OS independent access to the IP protocol family.

[0044] At the next level up, different ORBs use different low-level network transport protocols. For example, Sun's DOE systems uses ONC RPC, the Spring system uses an optimized sequenced packet protocol, UNO uses TCP/IP, some other vendors use DCE RPC, etc. However, while this may seem like fairly major difference it is actually comparatively easy to plug in different low level transport protocols.

[0045] Unfortunately, the different transport protocols also come with different data formats for simple data types. For example, integer values may have to be transmitted in big-endian byte order for an ONC ORB and in little-endian byte order for a DCE ORB. This affects the way that one can marshal and unmarshal arguments from the marshal buffers.

[0046] At the next level up, even if two ORBs agree on a standard format for simple data types, they may disagree on how to handle the IDL structured data types. For example both the Spring ORB and the DOE ORB use the ONC XDR format for simple data types, but when they transmit an array descriptor the DOE ORB simply transmits an integer specifying the length, whereas the Spring ORB transmits two integers, one specifying the array's lower bound and the other specifying the array's upper bound. This means that if one wants to have stubs that can be used between different ORBs then the stubs can't directly marshal things like array descriptors, but instead must call into some ORB specific code.

[0047] Finally there are likely to be different formats for the various kinds of IDL related meta-data, such as object references, method identifiers, exception identifiers, and type identifiers. For example, Spring uses integers as method identifiers. Other ORBs send either a simple method name of a fully qualified interface name plus method name combination.

The portability strategy

[0048] In the preferred embodiment, the general strategy has been to create stubs that are ORB independent and to conceal the ORB dependencies inside the individual object references. This has the major advantage that a single Java stub compiler can be used that can generate stubs that can be used with any ORB. However this means that there must be interfaces between the stubs and the object reference that allow the stubs to marshal arguments and invoke remote operations in an ORB independent manner.

[0049] Experience with using the *subcontract* mechanism in the Spring system was used in designing the separation between the stubs and the ORB specific layer. Spring permits different object references to have different formats and to have different invocation mechanisms, so as to be able to support things like replication and data caching. It does this by associating a software module called a *subcontract* with each object reference. When the Spring stubs want to marshal or invoke an object reference, they call into the subcontract associated with the object reference, so that the marshalling or object invocation is performed in a way that is appropriate to that subcontract.

[0050] In the JAVA ORB implementation of the preferred embodiment one extra abstract interface was added so that a single set of stubs could communicate with multiple ORBs. An abstract MarshalBuffer interface

was added and the creation of MarshalBuffers was moved from the stubs into the subcontracts, so that different ORBs could provide different sub-classes of MarshalBuffer which marshalled and unmarshalled data in the correct format for that ORB. Figure 3 provides an illustration of the Java ORB classes. In Figure 3, A Java-Application 70 uses a set of stubs 68 to talk to a set of remote objects (not shown). These stubs and applications comprise an "ORB Independent" layer 69 which interface generically with the "OS Independent/ORB dependent" layer 61. An exemplary "OS Independent/ ORB dependent" layer 61 comprises, for example, two different Spring subcontract mechanisms 66, 64 that use the same MarshalBuffer 60 and the same Spring network protocol code mechanism 56 to talk to the remote object (not shown). An ORB dependent code mechanism 62 for use by Sun's DOE ORB is shown, which uses a DOE MarshalBuffer 58 which in turn uses the DOE network protocol code mechanism 54. In this example, both the Spring network protocol code mechanism 56 and the DOE network protocol code mechanism 54 use the Java network "Socket" code mechanism 52. (The Java "Socket" class provides access to TCP/IP and UDP functionality in a way that is broadly similar to the sockets interfaces provided in the Berkeley UNIX distributions.). This Java network "Socket" code mechanism 52 uses Operating System (OS) services 51 from whatever OS it is running on.

The MarshalBuffer interface

[0051] In the preferred embodiment the interface between the stubs and the MarshalBuffer interface was defined so that the stubs not only marshal simple primitive types such as char and long, but are also given control over how structured IDL data types were marshalled. This meant providing generic MarshalBuffer methods for marshalling and unmarshalling array headers and union discriminators.

Putting the pieces together

[0052] In the context of the present invention different applications may call objects or send data to objects which have implementations that are associated with different ORBs but in this case using ORB-independent code/subcontract mechanisms to determine the target ORB, to find a MarshalBuffer that knows how to marshal the data for the target implementation and to communicate with the machine containing the target implementation. Referring now to Figure 5. the client application 112 again issues the call on stub 112. In this instance however, the stub 112 sends the call to an ORB-specific code/subcontract mechanism 212 determined in the preferred embodiment by an indication in the objref. (An alternative embodiment would include some ORB-ID mechanism for identifying the ORB-specific code mechanism required by a specific object call, where this ORB-ID mechanism might be specified when the object implementation is created and used each time this object is called thereafter. Those skilled in the art will recognize that there are many ways to identify the ORB-specific code required by an Object reference.) This ORB-specific code/subcontract mechanism 212 determines what format is required by the target ORB and provides a MarshalBuffer Object 210 capable of doing the correct marshaling, notifying the client stub 114 of which MarshalBuffer Object 210 is to be used. The client stub 114, using this MarshalBuffer Object 210 marshals the data and sends it to the network software code mechanism/device 116 as before. On the server side 118 the target ORB-specific code mechanism 214 knows how to unmarshal the data and passes it to the Unmarshal Buffer 216 which in the case of a Java transmission may be a virtual machine to interpret the byte-code data for execution by the server machine.

[0053] In the preferred embodiment, for any IDL object reference of type FOO, we provide a Java stub class FOO that consists of a set of stub methods and pointer to a subcontract object that contains information identifying the server object. Each object of the stub class will point to a different subcontract object, and these subcontract objects may have different implementations, allowing them to talk to different ORBS. When the stub methods wish to make an object invocation they ask the subcontract to give them a suitable MarshalBuffer object and then use that MarshalBuffer to marshal the arguments and unmarshal the results. An exemplary MarshalBuffer is shown in Figure 6. The MarshalBuffer interface shown in Figure 6 represents a clean separation between the functionality of the ORB dependent code in marshaling and unmarshalling data. The stubs understand the particular set of arguments or results that arc required for a particular IDL call. The stubs then call into the ORB dependent code mechanism that implements the MarshalBuffer interface in order to marshal (or unmarshal) each data item contained in the arguments or results. The ORB dependent code mechanism knows nothing about the IDL interface but simply marshals each data item in the correct format for that target ORB. Key concepts in the preferred embodiment are that (I) the MarshalBuffer interface such as shown in Figure 6 is an interface which different ORBs can provide their own implementation for, and (2) the ORB dependent code provides the MarshalBuffer object to the stubs.

[0054] So for example, referring now to Figure 7 an object reference of IDL type FOO 404 that points at a Spring server (not illustrated) might use Spring's Singleton subcontract 408. When the stubs for FOO 406 come to make a call on one of the FOO methods they first ask the subcontract 408 to give them a MarshalBuffer. The Singleton subcontract 408 will return a Spring MarshalBuffer 410 that will obey the Spring on-the-wire data formats. The stubs 406 then marshal the method arguments into that marshal buffer 410. After the stubs 406 have marshalled all the arguments, they call into the

Subcontract 408 to actually transmit the method invocation to the server. The Singleton subcontract 408 uses the Spring network protocol handlers 412 to transmit the request to the Spring server and get the results. The stubs 406 can them unmarshal the results from the MarshalBuffer 410 and return them to the client application 402.

[0055] In the preferred embodiment, a stub compiler *contojava* that generates complete Java client stubs for IDL interfaces is used.

[0056] In addition, a working Java ORB implementation that can communicate with both DOE and Spring servers is used. The code for talking to Spring includes two subcontracts (Caching and Singleton) and a Java implementation of Spring's proxy-proxy protocol. The code for talking to DOE includes a single subcontract (for the Basic Object Adapter (BOA)) and code for locating and activating DOE BOA objects. All of this ORB code is written in Java and is portable between different Java environments. It is believed that this ORB core could be easily extended with subcontracts that could talk to UNO or to ORBs implemented by other vendors.

Claims

1. A computer system having a processor (3), a memory (4), a display device (6), an input/output device (2), and an operating system (OS) for processing program code mechanism (10) invocations which are directed to one of a multiplicity of remote Object Request Brokers (ORBS), **characterised in that** the computer system further comprises:

    an ORB independent layer of code mechanisms (69) including program applications and related stubs; and
    an ORB dependent layer of code mechanisms (61), coupled to the ORB independent layer, the ORB dependent layer comprising

        program code mechanisms which are ORB-specific (212) and which are configured to generate a marshal code mechanism (210) to marshal data in an ORB required format, and
        program code mechanisms configured to use a specific network protocol code (116) whereby a program application code mechanism from the ORB independent layer can invoke a call on a remote implementation program code mechanism and the ORB-specific program code mechanisms in the ORB dependent layer (61) will determine an appropriate marshal code mechanism to marshal data in an ORB required format;
        thereby providing a computer system capable of communicating with a multiplicity of ORBs wherein each ORB requires data to be in a specific format.

2. A computer system as defined in claim 1, wherein the ORB independent layer of code mechanisms (69) including program applications and related stubs comprise object applications and related stubs.

3. A computer system as defined in claim 1 or 2, wherein the marshal code mechanism (210) to marshal data in an ORB required format which is located in the ORB-dependent layer (61) is a MarshalBuffer Object configured to execute operations to marshal data in an ORB specific format.

4. A computer system as defined in claim 1, 2 or 3, wherein the program code mechanism configured to use a specific network protocol code (116) communicates with a server computer (118) comprising:

    a receiving program code mechanism (120) comprising ORB dependent code configured to receive an object invocation from a remote computer system (110); and
    an ORB independent server application whereby the receiving program code mechanism calls into the ORB independent server application (214), passing in a MarshalBuffer that allows the server application to unmarshal arguments and marshal results without having to know which ORB a call came from.

5. A computer system a defined in claim 1,2,3 or 4, wherein the ORB-dependent layer (61) is also independent of the Operating System (OS) layer.

6. A method of operating a computer system having a processor (3), a memory (4), a display (6), an imput/output mechanism (2), an operating system and at least one client application program (112), one or more stub programs (114) related to the client application, the method performed by the computer being **characterised by** the steps of:

    invoking a call on a program implementation code mechanism, the call invocation being made by a client application wherein the call includes a reference to the program implementation code mechanism;
    using a stub program code mechanism which is related to the client application to receive the call invocation, wherein the stub program code mechanism has no knowledge of a format required for marshaling data provided by the client application in connection with the call invocation;

calling a first specific program code mechanism (212) by the stub program code mechanism, requesting the first specific program code mechanism (212) to provide a MarshalBuffer code mechanism (210) that knows how to format data provided by the client application in connection with the call invocation;

marshaling the data provided by the client application in connection with the call invocation, the marshaling being done by the stub program code mechanism using the MarshalBuffer code mechanism; and

sending the call invocation to a server (118) containing the program implementation code mechanism which is the target of the call.

7. The method described in claim 6, comprising the additional step of unmarshaling (216) results from the MarshalBuffer, the results supplied by the program implementation code mechanism which was called by the invocation.

8. The method described in claim 6 or 7, wherein the first specific program code mechanism (212) called by the stub program code mechanism is an Object Request Broker (ORB) specific code mechanism.

9. The method described in claim 6,7 or 8, wherein the stub program code mechanism which calls the ORB specific code mechanism (212) has no knowledge of the ORB which will process the call.

10. The method described in claim 6, 7, 8 or 9, wherein the Object Request Broker (ORB) is an Object Management Group (OMG) Common Object Request Broker (CORBA) compliant ORB.

11. The method of claim 6, 7, 8, 9 or 10, comprising the additional steps of:

receiving an object invocation from a remote computer system (110) by a server-side ORB dependent code mechanism (120); and

issuing a call into an ORB independent server application (214), the call made by the server-side ORB dependent code mechanism which passes in a MarshalBuffer that allows the server application to unmarshal arguments and marshal results without having to know which ORB a call came from.

12. A computer program product comprising a computer system usable storage medium (9) having computer readable code embodied therein for causing a computer system to process program code mechanism invocations which are directed to one of a multiplicity of remote Object Request brokers (ORBs), the computer program product being **characterised by**:

a first computer readable program code mechanism configured to comprise an ORB-independent layer (69) of code mechanisms including client applications (112) and related stub program code mechanisms (114); and

a second computer readable program code mechanism configured to comprise an ORB dependent layer (61) of code mechanisms coupled to the ORB independent layer (69), the ORB dependent layer comprising program code mechanisms which are ORB-specific and which are configured to generate a marshal code mechanism (210) to marshal data in an ORB required format and program code mechanisms (116) configured to use a specific network protocol code whereby a program application code mechanism from the ORB independent layer can invoke a call on a remote implementation program code mechanism (120) and the ORB-specific program code mechanisms in the ORB dependent layer will determine an appropriate marshal code mechanism to marshal data in an ORB required format, thereby providing a computer system capable of communicating with a multiplicity of ORBs wherein each ORB requires data to be in a specific format.

13. A computer product as defined in claim 12, wherein the ORB independent layer (69) of code mechanisms including program applications (112) and related stubs (114) comprise object applications and related stubs.

14. A computer program product as defined in claim 12 or 13, wherein the marshal code mechanism to marshal data in an ORB required format which is located in the ORB-dependent layer (61) is a MarshalBuffer Object (210) configured to execute operations to marshal data in an ORB specific format.

15. A computer program product as defined in claim 12, 13 or 14, wherein the program code mechanism configured to use a specific network protocol code (116) which is located in the ORB-dependent layer (61) is configured to use a network protocol code mechanism seleced from a group consisting of Spring network protocol code and DOE network protocol code.

16. A computer program product as defined in claim 12, 13, 14 or 15, wherein the marshal code mechanism to marshal data in an ORB required format which is located in the ORB- dependent layer (61) is also configured to unmarshal results returned to it.

Patentansprüche

1. Computersystem mit einem Prozessor (3), einem Speicher (4), einem Anzeigegerät (6), einem Ein-Ausgabegerät (2) und einem Betriebssystem (OS) zum Verarbeiten von Aufrufen von Programmcodemechanismen (10), die an eine Vielzahl von ortsfernen Object Request Brokern (ORBs) gerichtet sind, **dadurch gekennzeichnet, dass** das Computersystem ferner folgendes umfasst:

   eine ORB-unabhängige Schicht von Codemechanismen (69) einschließlich Programmanwendungen und zugehörigen Stubs; und
   eine ORB-abhängige Schicht von Codemechanismen (61), die mit der ORB-unabhängigen Schicht gekoppelt ist, wobei die ORB-abhängige Schicht folgendes umfasst:

      Programmcodemechanismen, die ORB-spezifisch (212) und so konfiguriert sind, dass sie einen Marshal-Codemechanismus (210) zum Marshalieren (Einsortieren) von Daten in einem von dem ORB benötigten Format erzeugen, und
      Programmcodemechanismen, die so konfiguriert sind, dass sie einen spezifischen Netzwerkprotokollcode (116) verwenden, so dass ein
      Programmanwendungscode-Mechanismus von der ORB-unabhängigen Schicht einen Aufruf an einen ortsfernen Implementationsprogrammcode-Mechanismus einleiten kann, und die ORB-spezifischen Programmcodemechanismen in der ORB-abhängigen Schicht (61) einen geeigneten Marshal-Codemechanismus ermitteln, um Daten in einem vorn ORB benötigten Format zu marschalieren;
      so dass ein Computersystem entsteht, das mit einer Vielzahl von ORBs kommunizieren kann, wobei jeder ORB ein spezielles Datenformat verlangt.

2. Computersystem nach Anspruch 1, bei dem die ORB-unabhängige Schicht von Codemechanismen (69) mit Programmanwendungen und zugehörigen Stubs Objektanwendungen und zugehörige Stubs umfassen.

3. Computersystem nach Anspruch 1 oder 2, bei dem der Marshal-Codemechanismus (210) zum Marshalieren von Daten in einem vom ORB benötigten Format, der sich in der ORB-abhängigen Schicht (61) befindet, ein MarshalBuffer-Objekt ist, das so konfiguriert ist, dass es Vorgänge zum Marshalieren von Daten in einem ORB-spezifischen Format ausführt.

4. Computersystem nach Anspruch 1, 2 oder 3, bei dem der ProgrammcodeMechanismus, der so konfiguriert ist, dass er einen spezifischen Netzwerkprotokollcode (116) verwendet, mit einem Servercomputer (118) kommuniziert, der folgendes umfasst:

   einen Empfangsprogrammcode-Mechanismus (120), der ORB-abhängigen Code umfasst, der so konfiguriert ist, dass er einen Objektaufruf von einem ortsfernen Computersystem (110) empfängt; und
   eine ORB-unabhängige Serveranwendung, so dass der Empfangsprogrammcode-Mechanismus in die ORB-unabhängige Serveranwendung (214) hineinruft und einen MarshalBuffer einbringt, so dass die Serveranwendung Argumente zurückmarshalieren (aussortieren) und Ergebnisse marshalieren kann, ohne wissen zu müssen, von welchem ORB ein Anruf stammte.

5. Computersystem nach Anspruch 1, 2, 3 oder 4, bei dem die ORB-abhängige Schicht (61) auch von der Betriebssystem-Schicht (OS) unabhängig ist.

6. Verfahren zum Betreiben eines Computersystems mit einem Prozessor (3), einem Speicher (4), einem Anzeigegerät (6), einem Ein-Ausgabemechanismus (2), einem Betriebssystem und wenigstens einem Client-Anwenderprogramm (112), einem oder mehreren Stub-Programmen (114) in Bezug auf die Client-Anwendung, wobei das mit dem Computer durchgeführte Verfahren durch die folgenden Schritte **gekennzeichnet** ist:

   Einleiten eines Aufrufs an einen Programmimplementationscode-Mechanismus, wobei der Aufruf durch eine Client-Anwendung erfolgt, wobei der Aufruf eine Referenz auf den Programmimplementationscode-Mechanismus beinhaltet;
   Verwenden eines Stubprogrammcode-Mechanismus, der sich auf die Client-Anwendung zum Empfangen des Aufrufs bezieht, wobei der Stubprogrammcode-Mechanismus das Format nicht kennt, das zum Marshalieren von Daten benötigt wird, die von der Client-Anwendung in Verbindung mit der Aufrufeinleitung bereitgestellt werden;
   Aufrufen eines ersten spezifischen Programmcodemechanismus (212) durch den Stubprogrammcode-Mechanismus, Anfordern des ersten spezifischen Programmcodemechanismus (212) zur Bereitstellung eines MarshalBuffer-Codemechanismus (210), der weiß, wie von der Client-Anwendung bereitgestellte Daten in Verbindung mit dem Aufruf formatiert werden;

Marshalieren der von der Client-Anwendung bereitgestellten Daten in Verbindung mit dem Aufruf, wobei die Marshalierung vom Stubprogrammcode-Mechanismus mit Hilfe des MarshalBuffer-Codemechanismus erfolgt; und

Senden des Aufrufs zu einem Server (118), der den Programmimplementationscode-Mechanismus enthält, der das Ziel des Rufs ist.

7. Verfahren nach Anspruch 6, umfassend den zusätzlichen Schritt des Zurückmarshalierens (216) von Ergebnissen vom MarshalBuffer, wobei die Ergebnisse vom Programmimplementationscode-Mechanismus geliefert werden, der mit dem Aufruf gerufen wurde.

8. Verfahren nach Anspruch 6 oder 7, bei dem der erste spezifische ProgrammcodeMechanismus (212), der vom Stubprogrammcode-Mechanismus gerufen wurde, ein für einen Object Request Broker (ORB) spezifischer Codemechanismus ist.

9. Verfahren nach Anspruch 6, 7 oder 8, bei dem der Stubprogrammcode-Mechanismus, der den ORB-spezifischen Codemechanismus (212) ruft, den ORB nicht kennt, der den Ruf verarbeitet.

10. Verfahren nach Anspruch 6, 7, 8 oder 9, bei dem der Objekt Request Broker (ORB) ein ORB ist, der mit einem Common Object Request Broker (CORBA) gemäß der Object Management Group (OMG) kompatibel ist.

11. Verfahren nach Anspruch 6, 7, 8, 9 oder 10, umfassend die folgenden zusätzlichen Schritte:

Empfangen eines Objektaufrufs von einem ortsfernen Computersystem (110) auf einem serverseitigen ORB-abhängigen Codemechanismus (120); und

Ausgeben eines Rufs in eine ORB-unabhängige Serveranwendung (214), wobei der Ruf durch den serverseitigen ORB-abhängigen Codemechanismus erfolgt, der einen MarshalBuffer einbringt, der es zulässt, dass die Serveranwendung Argumente zurückmarshaliert und Ergebnisse marshaliert, ohne wissen zu müssen, von welchem ORB ein Ruf stammt.

12. Computerprogrammprodukt, umfassend ein in einem Computersystem benutzbares Speichermedium (9), in dem rechnerlesbarer Code eingebettet ist, mit dem veranlasst werden kann, dass ein Computersystem Programmcodemechanismus-Aufrufe verarbeitet; die an einen aus einer Vielzahl von ortsfernen Object Request Brokern (ORBs) gerichtet sind, wobei das Computerprogrammprodukt **gekennzeichnet ist durch**:

einen ersten rechnerlesbaren Programmcodemechanismus, der so konfiguriert ist, dass er eine ORB-unabhängige Schicht (69) von Codemechanismen umfasst, einschließlich Client-Anwendungen (112) und zugehörigen Stubprogrammcode-Mechanismen (114); und

einen zweiten rechnerlesbaren Programmcodemechanismus, der so konfiguriert ist, dass er eine ORB-abhängige Schicht (61) von Codemechanismen umfasst, die mit der ORB-unabhängigen Schicht (69) gekoppelt sind, wobei die ORB-abhängige Schicht Programmcodemechanismen umfasst, die ORB-spezifisch und so konfiguriert sind, dass sie einen Marshal-Codemechanismus (210) zum Marshalieren von Daten in einem vom ORB benötigten Format erzeugen, und Programmcodemechanismen (116), die so konfiguriert sind, dass sie einen spezifischen Netzwerkprotokollcode benutzen, so dass ein Programmanwendungscode-Mechanismus von der ORB-unabhängigen Schicht einen Ruf auf einem ortsfernen Implementationsprogrammcodemechanismus (120) einleiten kann, und die ORB-spezifischen Programmcodemechanismen in der ORB-abhängigen Schicht einen geeigneten Marshal-Codemechanismus zum Marshalieren von Daten in einem vom ORB benötigten Format bestimmen, so dass ein Computersystem bereitgestellt wird, das in der Lage ist, mit einer Vielzahl von ORBs zu kommunizieren, wobei jeder ORB Daten in einem bestimmten Format verlangt.

13. Computerprodukt nach Anspruch 12, bei dem die ORB-unabhängige Schicht (69) von Codemechanismen mit Programmanwendungen (112) und zugehörigen Stubs (114) Objektanwendungen und zugehörige Stubs umfassen.

14. Computerprogrammprodukt nach Anspruch 12 oder 13, bei dem der Marshal-Codemechanismus zum Marshalieren von Daten in einem vom ORB benötigten Format, der sich in der ORB-abhängigen Schicht (61) befindet, ein MarshalBuffer Object (210) ist, das so konfiguriert ist, dass es Vorgänge zum Marshalieren von Daten in einem ORB-spezifischen Format ausführt.

15. Computerprogrammprodukt nach Anspruch 12, 13 oder 14, bei dem der Programmcodemechanismus, der so konfiguriert ist, dass er einen spezifischen Netzwerkprotokollcode (116) benutzt, der sich in der ORB-abhängigen Schicht (61) befindet, so konfiguriert ist, dass er einen Netzwerkprotokollcode-Mechanismus benutzt, der ausgewählt ist aus einer Gruppe bestehend aus Spring-Netzwerkprotokollcode und DOE-Netzwerkprotokollcode.

16. Computerprogrammprodukt nach Anspruch 12, 13, 14 oder 15, bei dem der Marshal-Codemechanismus zum Marshalieren von Daten in einem vom ORB benötigten Format, der sich in der ORB-abhängigen Schicht (61) befindet, ebenfalls konfiguriert ist, um an ihn zurückgegebene Ergebnisse zurückzumarshalieren.

Revendications

1. Système d'ordinateur ayant un processeur (3), une mémoire (4), un dispositif d'affichage (6), un dispositif d'entrée/sortie (2), et un système d'exploitation (OS) pour traiter des invocations de mécanismes (10) de code de programmation qui sont orientées vers l'un d'une multiplicité d'Object Request Brokers (Courtiers de Requêtes) (ORB) à distance, **caractérisé en ce que** le système d'ordinateur comprend en outre :

    une couche indépendante des ORB de mécanismes de code (69) incluant des applications de programmation et des souches associées; et
    une couche dépendante des ORB de mécanismes de code (61), couplée à la couche indépendante des ORB, la couche dépendante des ORB comprenant

        des mécanismes de code de programmation qui sont spécifiques aux ORB (212) et qui sont configurés pour générer un mécanisme de code d'assemblage (210) pour assembler des données dans un format requis par un ORB, et
        des mécanismes de code de programmation configurés pour utiliser un code spécifique de protocole réseau (116) par lesquels un mécanisme de code d'application de programmation de la couche indépendante des ORB peut invoquer un appel sur un mécanisme de code de programme d'implémentation à distance et les mécanismes de code de programmation spécifiques aux ORB dans la couche dépendante des ORB (61) détermineront un mécanisme de code d'assemblage approprié pour assembler des données dans un format requis par un ORB;
        foumissant ainsi un système d'ordinateur capable de communiquer avec une multiplicité d'ORB dans lequel chaque ORB exige que les données soient dans un format spécifique.

2. Système d'ordinateur tel que défini dans la revendication 1, dans lequel la couche indépendante des ORB de mécanismes de code (69) incluant des applications de programmation et des souches associées comprennent des applications objets et souches associées.

3. Système d'ordinateur tel que défini dans la revendication 1 ou 2, dans lequel le mécanisme de code d'assemblage (210) pour assembler des données dans un format requis par un ORB qui se trouve dans la couche dépendante des ORB (61) est un MarshalBuffer Object configuré pour exécuter des opérations pour assembler des données dans un format spécifique à un ORB.

4. Système d'ordinateur tel que défini dans la revendication 1, 2 ou 3, dans lequel le mécanisme de code de programmation configuré pour utiliser un code spécifique de protocole réseau (116) communique avec un ordinateur serveur (118) comprenant:

    un mécanisme de code de programmation récepteur (120) comprenant un code dépendant des ORB configuré pour recevoir une invocation d'objet d'un système d'ordinateur à distance (110); et
    une application serveur indépendante des ORB par laquelle le mécanisme de code de programmation récepteur appelle l'application serveur indépendante des ORB (214), passant dans un MarshalBuffer qui autorise l'application serveur à désassembler les arguments et à assembler les résultats sans avoir à savoir de quel ORB un appel est venu.

5. Système d'ordinateur tel que défini dans la revendication 1, 2, 3 ou 4, dans lequel la couche dépendante des ORB (61) est aussi indépendante de la couche du système d'exploitation (OS).

6. Méthode d'exploitation d'un système ordinateur ayant un processeur (3), une mémoire (4), un affichage (6), un mécanisme d'entrée/sortie (2), un système d'exploitation et au moins un programme d'application cliente (112), un ou plusieurs programmes souches (114) associés à l'application cliente, la méthode exécutée par l'ordinateur étant **caractérisée par** les étapes consistant à :

    invoquer un appel sur un mécanisme de code d'implémentation de programme, l'invocation de l'appel étant faite par une application cliente où l'appel inclut une référence au mécanisme de code d'implémentation de programme;
    utiliser un mécanisme de code de programme souche qui est associé à l'application cliente pour recevoir l'invocation d'appel, où le mécanisme de code de programme souche n'a aucune connaissance d'un format requis pour as-

sembler les données fournies par l'application cliente en rapport à l'invocation de l'appel;
appeler un premier mécanisme de code spécifique de programmation (212) par le mécanisme de code de programme souche, demandant que le premier mécanisme de code spécifique de programmation (212) fournisse un mécanisme de code MarshalBuffer (210) qui sait comment formater les données fournies par l'application cliente en rapport avec l'invocation d'appel;
assembler les données fournies par l'application cliente en rapport avec l'invocation d'appel, l'assemblage étant fait par le mécanisme de code de programme souche en utilisant le mécanisme de code de MarshalBuffer; et
envoyer l'invocation d'appel à un serveur (118) contenant le mécanisme de code d'implémentation de programme qui est la cible de l'appel.

7. Méthode décrite dans la revendication 6, comprenant l'étape additionnelle consistant à désassembler (216) les résultats du MarshalBuffer, les résultats fournis par le mécanisme de code d'implémentation de programme qui a été appelé par l'invocation.

8. Méthode décrite dans la revendication 6 ou 7, dans laquelle le premier mécanisme de code spécifique de programmation (212) appelé par le mécanisme de code de programme souche est un mécanisme de code spécifique à un Object Request Broker (ORB).

9. Méthode décrite dans la revendication 6, 7 ou 8, dans laquelle le mécanisme de code de programme souche qui appelle le mécanisme de code spécifique à un ORB (212) n'a aucune connaissance de l'ORB qui traitera l'appel.

10. Méthode décrite dans la revendication 6, 7, 8 ou 9, dans laquelle l'Object Request Broker (ORB) est un ORB conforme au Common Object Request Broker (CORBA) du Object Management Group (OMG).

11. Méthode de la revendication 6, 7, 8, 9 ou 10, comprenant les étapes supplémentaires consistant à :

recevoir un invocation d'objet d'un système d'ordinateur à distance (110) par un mécanisme de code dépendant de l'ORB côté serveur (120); et
émettre un appel à une application serveur indépendante de l'ORB (214), l'appel fait par le mécanisme de code dépendant de l'ORB côté serveur qui passe dans un MarshalBuffer qui autorise l'application serveur à désassembler les arguments et à assembler les résultats sans avoir à savoir de quel ORB un appel est venu.

12. Produit de programme informatique comprenant un support de stockage utilisable par un système d'ordinateur (9) ayant un code lisible par ordinateur incorporé dans celui-ci pour faire qu'un système d'ordinateur traite des invocations de mécanismes de code de programmation qui sont orientées vers l'un d'une multiplicité d'Object Request Brokers (ORB) à distance, le produit de programme informatique étant **caractérisé par** :

un premier mécanisme de code de programmation lisible par ordinateur configuré pour comprendre une couche indépendante des ORB (69) de mécanismes de code incluant des applications clientes (112) et des mécanismes de code de programmes souches associés (114); et
un deuxième mécanisme de code de programmation lisible par ordinateur configuré pour comprendre une couche dépendante des ORB (61) de mécanismes de code couplée à la couche indépendante des ORB (69), la couche dépendante des ORB comprenant des mécanismes de code de programmation qui sont spécifiques aux ORB et qui sont configurés pour générer un mécanisme de code d'assemblage (210) pour assembler des données dans un format requis par un ORB et des mécanismes de code de programmation (116) configurés pour utiliser un code spécifique de protocole réseau en vertu de quoi un mécanisme de code d'application de programmation de la couche indépendante des ORB peut invoquer un appel sur un mécanisme de code de programme d'implémentation (120) à distance et les mécanismes de code de programmation spécifiques aux ORB dans la couche dépendante des ORB détermineront un mécanisme de code d'assemblage approprié pour assembler des données dans un format requis par un ORB, fournissant ainsi un système d'ordinateur capable de communiquer avec une multiplicité d'ORB dans lequel chaque ORB exige que les données soient dans un format spécifique.

13. Produit informatique tel que défini dans la revendication 12, dans lequel la couche indépendante des ORB (69) de mécanismes de code incluant des applications de programmation (112) et des souches associées (114) comprennent des applications objets et des souches associées.

14. Produit de programme informatique tel que défini dans la revendication 12 ou 13, dans lequel le mécanisme de code d'assemblage pour assembler des données dans un format requis par un ORB qui

se trouve dans la couche dépendante des ORB (61) est un MarshalBuffer Object (210) configuré pour exécuter des opérations pour assembler des données dans un format spécifique à un ORB.

15. Produit de programme informatique tel que défini dans la revendication 12, 13 ou 14, dans lequel le mécanisme de code de programmation configuré pour utiliser un code spécifique de protocole réseau (116) qui se trouve dans la couche dépendante des ORB (61) est configuré pour utiliser un mécanisme de code de protocole réseau sélectionné d'un groupe consistant en code de protocole réseau Spring et en code de protocole réseau DOE.

16. Produit de programme informatique tel que défini dans la revendication 12, 13, 14 ou 15, dans lequel le mécanisme de code d'assemblage pour assembler des données dans un format requis par un ORB qui se trouve dans la couche dépendante des ORB (61) est aussi configuré pour désassembler les résultats qui lui sont renvoyés.

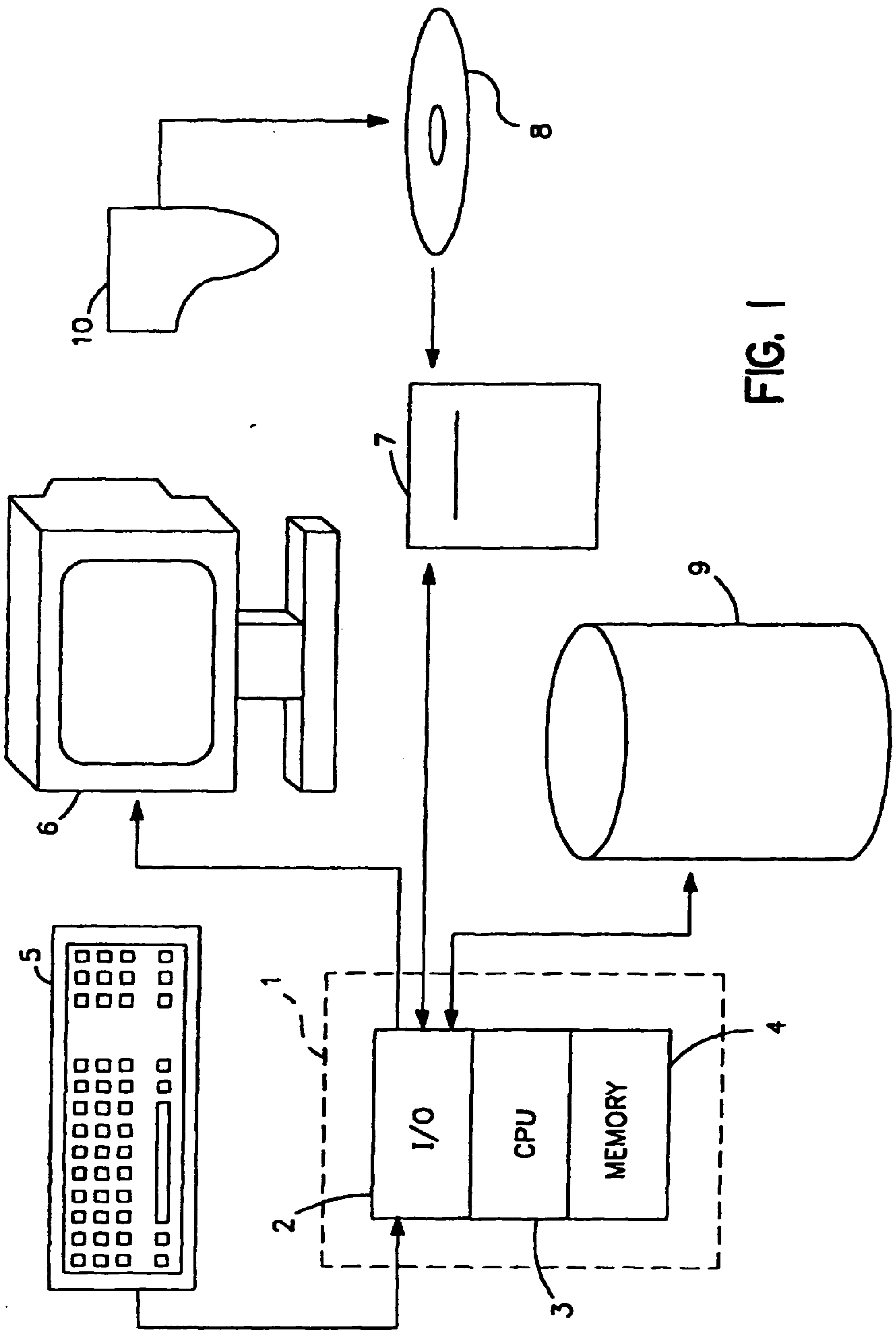

FIG. 1

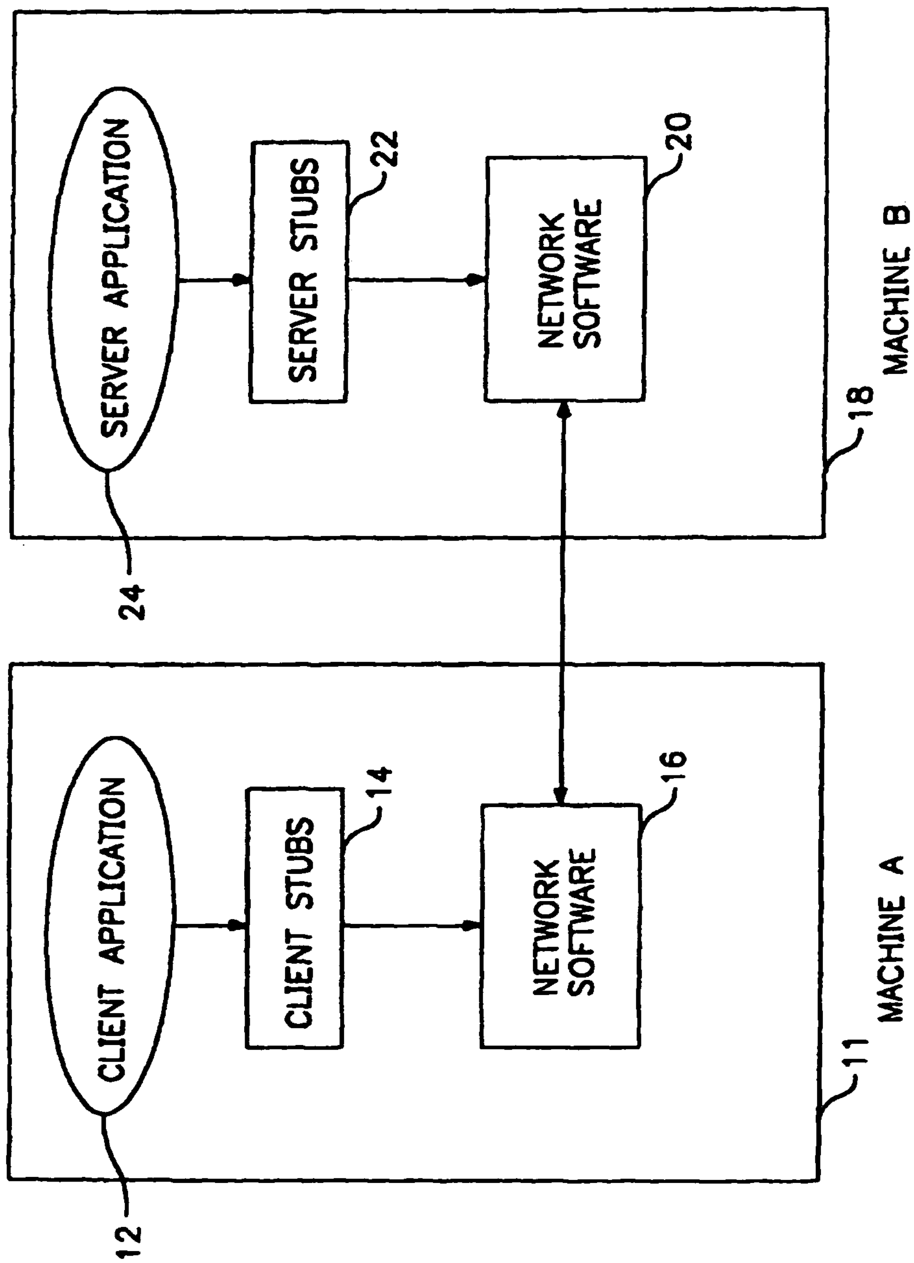

FIG. 2
(PRIOR ART)

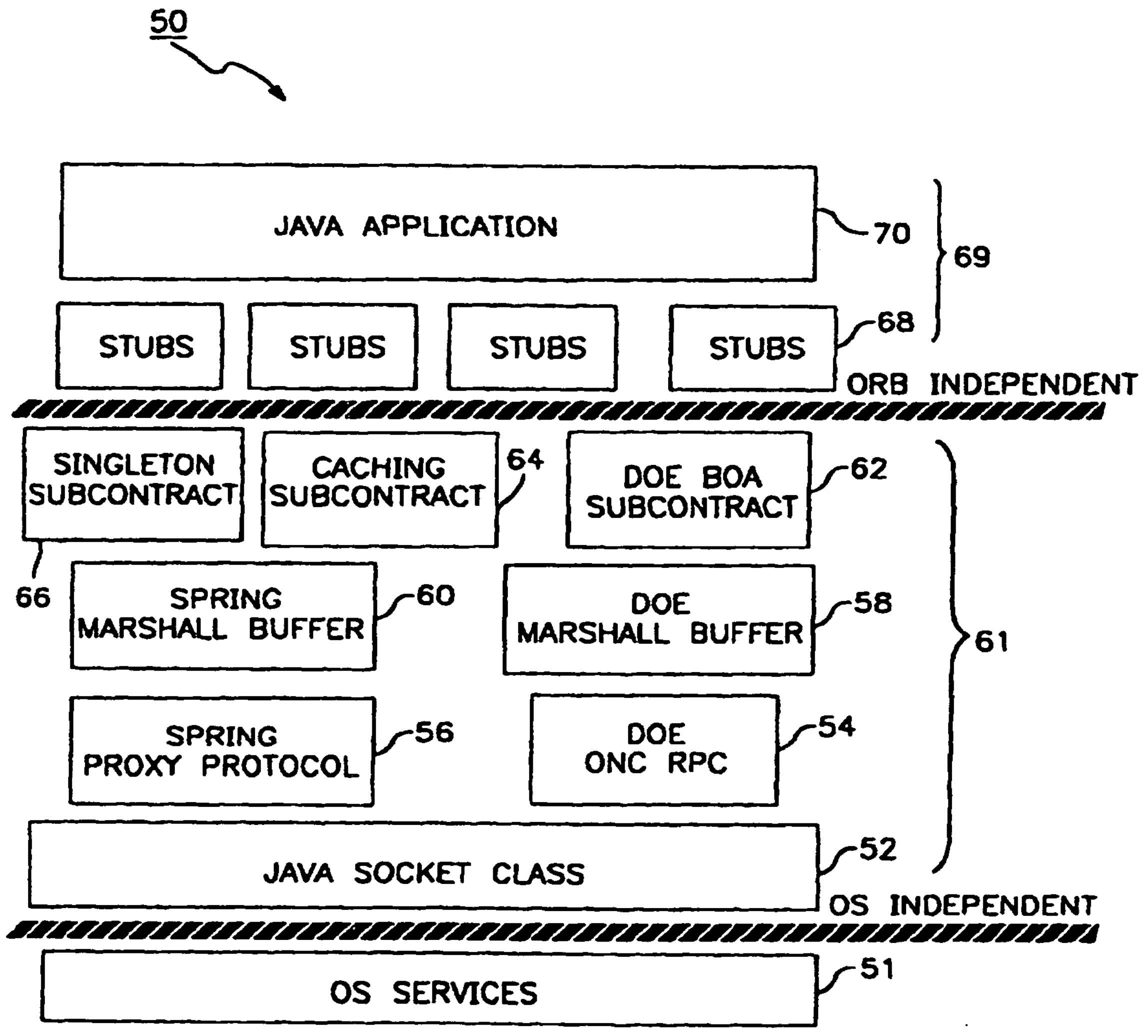
50
JAVA APPLICATION
70
69
STUBS
STUBS
STUBS
STUBS
68
ORB INDEPENDENT
SINGLETON SUBCONTRACT
CACHING SUBCONTRACT
64
DOE BOA SUBCONTRACT
62
66
SPRING MARSHALL BUFFER
60
DOE MARSHALL BUFFER
58
61
SPRING PROXY PROTOCOL
56
DOE ONC RPC
54
JAVA SOCKET CLASS
52
OS INDEPENDENT
OS SERVICES
51

FIG. 3

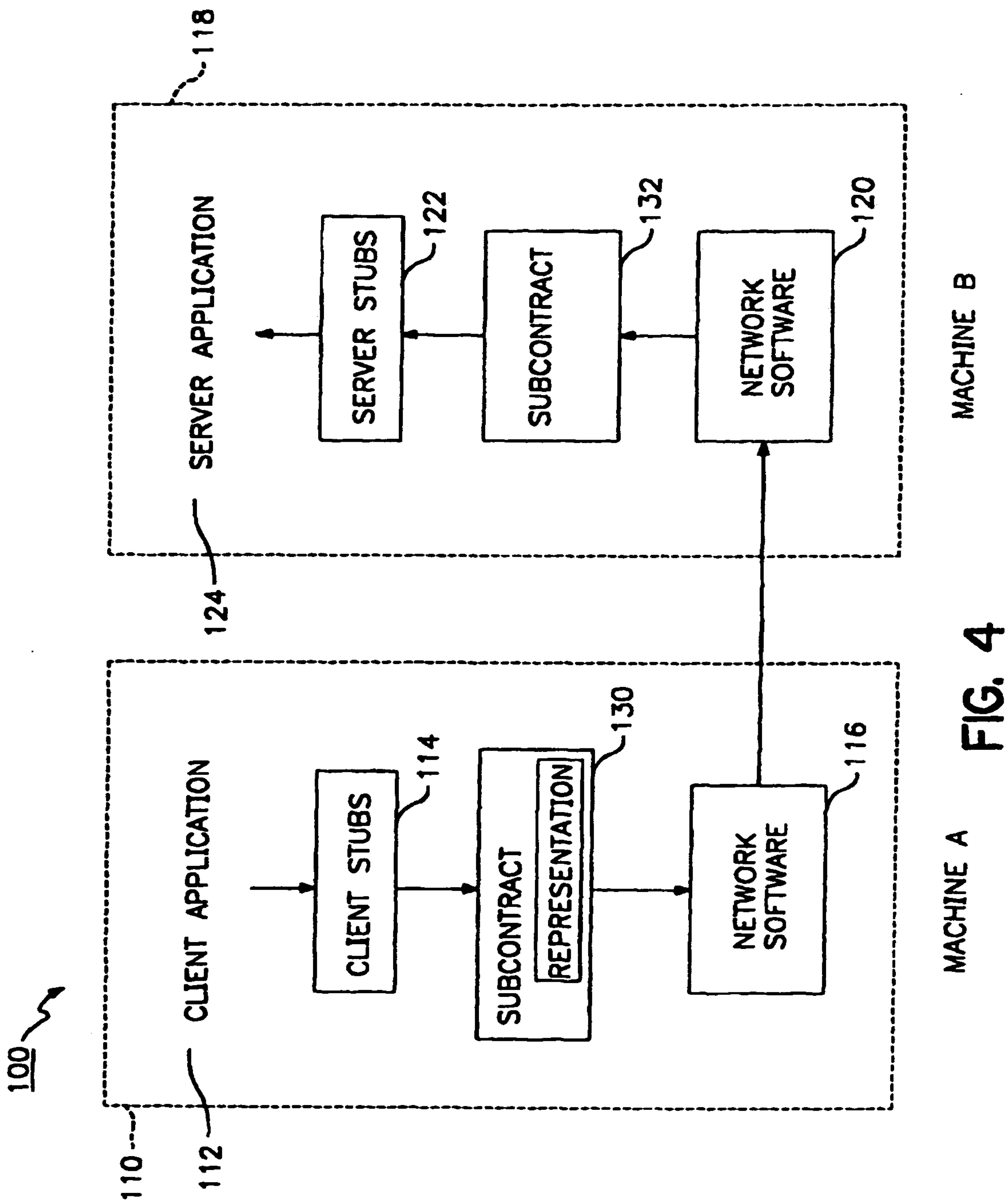
100
110
112
CLIENT APPLICATION
CLIENT STUBS
114
SUBCONTRACT
REPRESENTATION
130
NETWORK SOFTWARE
116
MACHINE A
118
124
SERVER APPLICATION
SERVER STUBS
122
SUBCONTRACT
132
NETWORK SOFTWARE
120
MACHINE B

FIG. 4

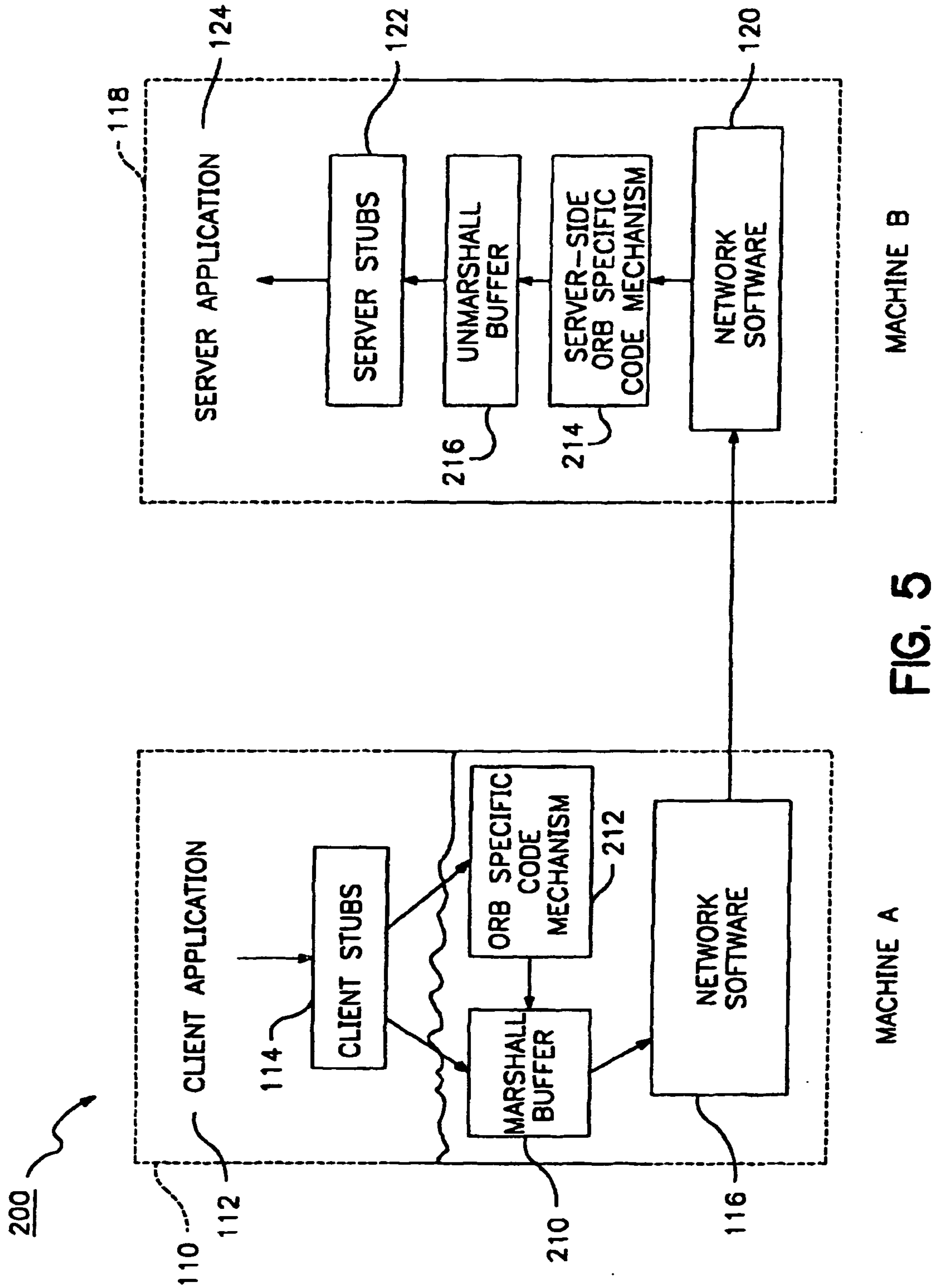
200
110
112
CLIENT APPLICATION
114
CLIENT STUBS
ORB SPECIFIC CODE MECHANISM
212
MARSHALL BUFFER
210
NETWORK SOFTWARE
116
MACHINE A
118
124
SERVER APPLICATION
122
SERVER STUBS
UNMARSHALL BUFFER
216
SERVER-SIDE ORB SPECIFIC CODE MECHANISM
214
120
NETWORK SOFTWARE
MACHINE B

FIG. 5

```
//
// MarshalBuffer describes a set of arguments or results to or from
// an IDL call
//
//package OP;

public interface MarshalBuffer {
    byte getByte () throws CORBA.SystemException;
    byte [] getbytes (int len) throws CORBA.SystemException;
    short getShort () throws CORBA.SystemException;
    int getInt () throws CORBA.SystemException;
    long getLong () throws CORBA.SystemException;
    char getChar () throws CORBA.SystemException;
    char [] getChars (int len) throws CORBA.SystemException;
    boolean getBool () throws CORBA.SystemException;
    String getString () throws CORBA.systemException;
    float getFloat () throws CORBA.SystemException;
    double getDouble () throws CORBA.SystemException;
    public OP.SequenceHeader getSequencePreamble () throws CORBA.SystemException;

    void putByte (byte X) throws CORBA.SystemException;
    void putBytes (byte x []) throws CORBA.SystemException;
    void putShort (short x) throws CORBA.SystemException;
    void putInt (int x) throws CORBA.SystemException;
    void putLong (long x) throws CORBA.SystemException;
    void putChar (char x) throws CORBA.SystemException;
    void putChars (char x []) throws CORBA.SystemException;
    void putString (String x) throws CORBA.SystemException;
    void putBool (boolean x) throws CORBA.SystemException;
    void putFloat (float x) throws CORBA.SystemException;
    void putDouble (double x) throws CORBA.SystemException;
    public void putSequencePreamble (int length) throws CORBA.SystemException;

    void putNull () throws CORBA.SystemException;
    void unmarshal_object (CORBA.CORBA_Object.Object ref) throws CORBA.SystemException;
    //Release means the MarshalBuffer can discard or recycle its contents.
    void release ();

    //reset means that the marshal buffer should be put back into the same
    //state as a newly constructed MarshalBufffer
    void reset () throws CORBA.SystemException;
```

FIG. 6

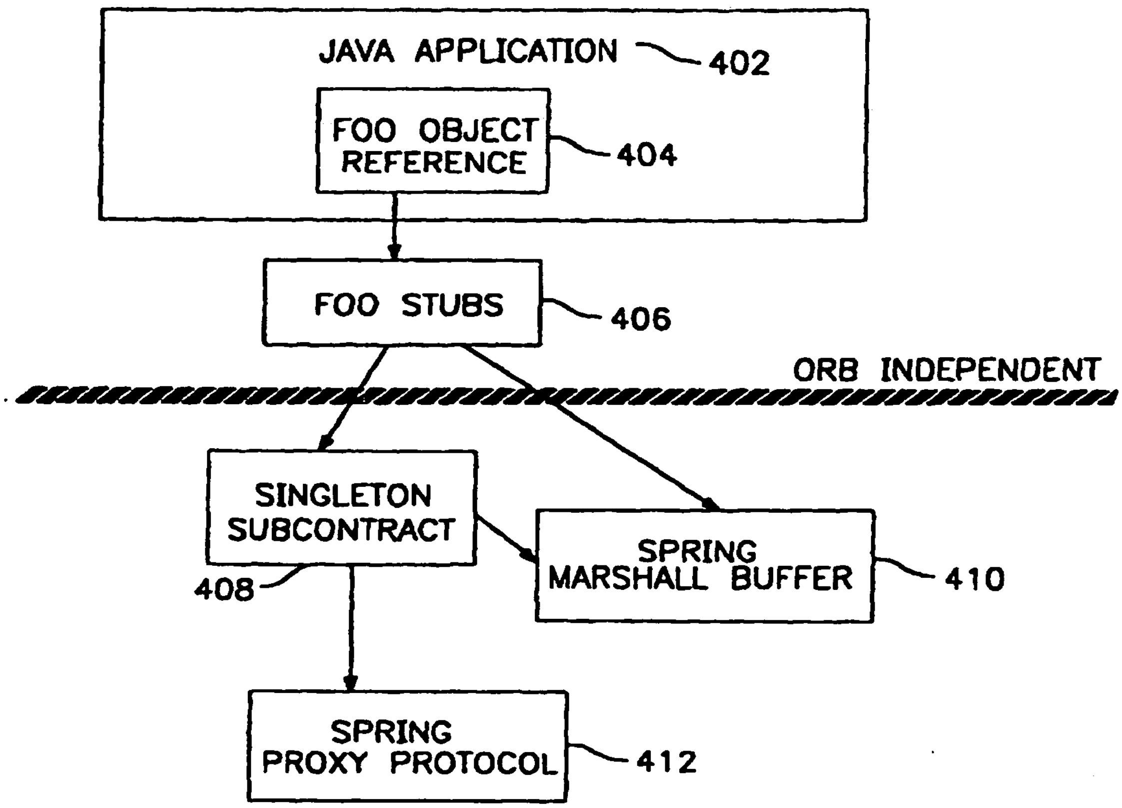
JAVA APPLICATION
402
FOO OBJECT REFERENCE
404
FOO STUBS
406
ORB INDEPENDENT
SINGLETON SUBCONTRACT
408
SPRING MARSHALL BUFFER
410
SPRING PROXY PROTOCOL
412

FIG. 7